ESD-TR-72-147, Vol. 2

MTR-2254, Vol. II

HARMONIOUS COOPERATION OF PROCESSES OPERATING ON A COMMON SET OF DATA, PART 2

D. Elliott Bell

DECEMBER 1972

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 671A
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-71-C-0002

ESD-TR-72-147, Vol. 2 MTR-2254, Vol. II

HARMONIOUS COOPERATION OF PROCESSES OPERATING ON A COMMON SET OF DATA, PART 2

D. Elliott Bell

DECEMBER 1972

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 671A
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-71-C-0002

FOREWORD

The work described in this report was carried out under the sponsorship of the Deputy for Command and Management Systems, Project 671A, by The MITRE Corporation, Bedford, Massachusetts, under Contract No. F19(628)-71-C-0002.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.

MELVIN B. EMMONS, Colonel, USAF
Director, Information Systems Technology
Deputy for Command and Management Systems

ABSTRACT

This report provides algorithms (and proofs of their correctness) which actualize the data-sharing model of "Harmonious Cooperation of Processes Operating on a Common Set of Data, Part 1" (MTR-2254). Also included are a sketch of the coordination of the algorithms in Part 2 in the operation of the Scheduler of Part 1 and a brief analysis of the storage required to implement this version of the Scheduler.

ACKNOWLEDGMENTS

I owe special thanks to Leonard J. LaPadula for his substantial contribution to this report. He not only wrote Part 1 of this series (the direct stimulus of this report) but also spent much time checking my work, answering my questions about his report and about the data-sharing task of Project 6710, and encouraging me in my preparation of this paper.

I am also greatly indebted to Mrs. Judith Clapp for her patient and meticulous examination of this report and for the many substantial improvements she suggested to me. Finally I want to thank Miss Dona Karas who typed this report. Her competence and her patience left nothing to be desired and made the final editing of this report almost effortless.

TABLE OF CONTENTS

		Page
LIST OF ILLUSTRATIONS		vi
SECTION I	INTRODUCTION	1
	THE INTENT OF THIS REPORT	1
	A SUMMARY OF THE SITUATION	1
	THE DIRECTION OF THE REPORT	2
SECTION II	ALGORITHM β	4
	INTRODUCTION	4
	NOTATION	4
	ALGORITHM β	5
	ANALYSIS OF THE ALGORITHM	5
SECTION III	UPPER TRIANGULAR MATRICES AND SAFE PERMUTATIONS	8
	INTRODUCTION	8
	NOTATION AND BASIC FACTS	8
	MAIN DEVELOPMENT	9
SECTION IV	ALGORITHM γ	14
	INTRODUCTION	14
	DEVELOPMENT	14
	ALGORITHM γ	17
	ANALYSIS OF THE ALGORITHM	17
SECTION V	A SCOREKEEPER FOR HARMONIOUS COOPERATION	19
	INTRODUCTION	19
	DESCRIPTION OF THE SCORECARD	19
	USE OF THE SCOREKEEPER	22
SECTION VI	AN OVERVIEW OF THE OPERATION OF THE SCHEDULER	37
	INTRODUCTION	37
	DESCRIPTION OF THE OPERATION OF THE SCHEDULER	37
	ANALYSIS OF STORAGE REQUIREMENTS	41
	CONCLUDING REMARKS	42
BIBLIOGRAPHY		44

LIST OF ILLUSTRATIONS

Figure Number		Page
1	The UVQJ Matrix	21
2	A Process Begins Its Run	24
3	The Investigation of a Write Mode Request	26
4	The Investigation of a Read Mode Request	28
5	The Investigation of an Inquiry-Use Mode Request	30
6	The Investigation of a Request for a Subelement	32
7	The Release of a Datum	33
8	P_i Releases a Subelement	33
9	The Reduction of a Claim List	34
10	A Process Finishes	34

SECTION I

INTRODUCTION

THE INTENT OF THIS REPORT

The harmonious cooperation guaranteed by Theorem 18 of "Harmonious Cooperation of Processes Operating on a Common Set of Data, Part 1" (called "HC1" in this report) depends on a Scheduler which evaluates each request by a process P_i for a datum S_j. The evaluation is carried out using strategy α to determine that granting a request would not cause conflict and would result in a safe situation. The latter condition will be satisfied if there is a safe permutation of the processes P by Theorems 5 and 7 of HC1. It is the intent of this report to provide concrete algorithms for making these determinations and for handling the record-keeping involved when a process releases data, reduces its claim lists, or begins or ends its run. It is always assumed that processes are acting "harmoniously": no error checks are included in this report.

A SUMMARY OF THE SITUATION

The focus throughout this report is a black box called a Scheduler which acts as a "librarian" for the data base $\underline{D} = (S,\underline{R})$. As in HC1, $S = \{S_1, \ldots, S_m\}$ is a set of elements, $\underline{R}$ is a binary relation on S and $\underline{R}'$ is a binary relation on the elements of elements of S. The processes $\underline{P} = \{P_1, \ldots, P_n\}$ make requests for elements of S. The Scheduler grants or rejects requests for elements of S so that

1) no conflict results,

2) deadlock never occurs, and

3) no process is permanently blocked.

THE DIRECTION OF THE REPORT

The approach of this report is to translate the determinations the Scheduler must make into mathematical propositions. The algorithms introduced provide an efficient way of translating the mathematical propositions into routinely decidable questions.

Section II addresses the question of whether the granting of a request will result in a safe situation. Using the approach of HCl, this determination is made by determining whether the resulting blocking graph of the processes $\underline{P}$ would be loop-free. Algorithm β allows one to determine whether the requesting process would be contained in a closed path in the blocking graph if its request were granted. This section utilizes digraph theory via adjacency matrices.

Section III provides rigorous justification for recording the potential blocking matrix in strict upper triangular form. Using matrix operations it is shown that the existence of a safe permutation of the processes $\underline{P}$ is equivalent to the existence of a strict upper triangular matrix representation of the potential blocking matrix.

Section IV provides an algorithm for generating a safe permutation of the processes when it is known that one exists. In addition,

it provides a method of altering the potential blocking matrix so that it will correspond to the current safe permutation A. Section IV utilizes matrix theory.

Section V describes a Scorecard for keeping track of the data elements S_j. In addition, the alteration of the Scorecard under the nine basic data operations is described. Diagrams giving capsule descriptions of the alterations of the Scorecard are included.

Section VI begins by describing how the various algorithms fit together to aid the Scheduler in its task. An analysis of the precise storage requirements for this implementation of the Scheduler are calculated for n processes using m pieces of shared data. For example, it is calculated that on a 256K word machine having 16-bit words, 64 concurrent users of 1000 files (each file having no more than 64K records) would require a storage overhead of 6.44% of the total storage capacity. The section ends with a short discussion of this report's functions and its omissions.

SECTION II

ALGORITHM β

INTRODUCTION

In this section, a mathematical algorithm is presented that will enable the Scheduler to determine whether the granting of a request will result in a safe permutation of the processes. If the request would not result in a safe permutation, then by definition the blocking graph of the processes after the request is granted will contain a closed path. If P is the requesting process, then it is clear that P would be contained in this closed path. Algorithm β provides a means of determining whether P would be contained in such a closed path.

Algorithm β is an outgrowth of digraph theory and the particular requirements of the data-sharing task of Project 6710. The blocking graph of the processes $\underline{P}$ is represented by its adjacency matrix. Algorithm β provides a method of determining from such an adjacency matrix whether a given vertex v_i is contained in a cycle in a digraph, thereby establishing whether the granting of the request will yield a safe permutation.

NOTATION

Let G be a directed graph with n vertices and let $\alpha = (\alpha_{ij})_{1 \le i,j \le n}$ be the associated 0-1 adjacency matrix: $\alpha_{ij} = 1$ if there is an arc

from vertex v_i to vertex v_j and $\alpha_{ij} = 0$ if there is no such arc. For $1 \leq i \leq n$, we let $\alpha_i = (\alpha_{i1}, \ldots, \alpha_{in})$ be the ith row of α.

Let r_1 and r_2 be 1 x n row vectors with $r_i = (r_{i1}, \ldots, r_{in})$ for i = 1,2. Let $[r_1, r_2] = (\max(r_{11}, r_{21}), \ldots, \max(r_{1n}, r_{2n}))$.

ALGORITHM β

Set $T = \alpha_i$ and $E = \varphi$, the empty set.

Step 1: If there is a $j \notin E$ such that T_j (the jth component of T) is not zero, replace T by $[T, \alpha_{j_0}]$, where j_0 is the least such j, and replace E by $E \cup \{j_0\}$.

Step 2: If $T_i = 1$, there is a circuit containing v_i: stop the algorithm. If $T_i = 0$, repeat step 1.

If it becomes impossible to perform step 1, then there is no cycle in G containing v_i.

ANALYSIS OF THE ALGORITHM

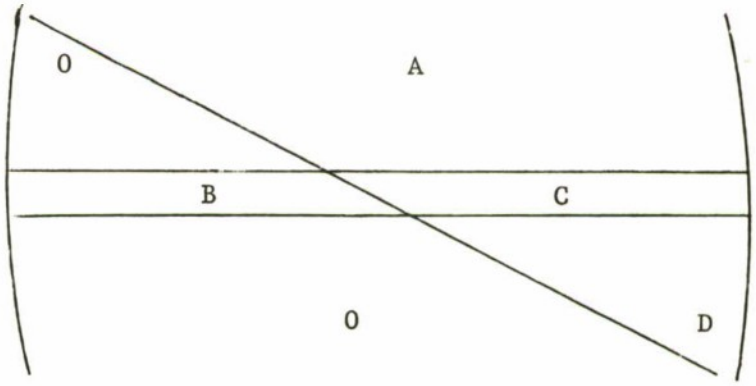
For $T^{(1)} = \alpha_i$, $T^{(1)}$ represents the vertices of G connected to v_i by paths of length 1. If $\{j_1, \ldots, j_k\}$ are the indices of the nonzero components of $T^{(1)}$, letting $T^{(2)} = [\ldots[[T^{(1)}, \alpha_{j_1}], \ldots], \alpha_{j_k}]$ gives a representation of the vertices of G that are connected to v_i by paths of length ≤ 2. Repeating this procedure, we will have $T^{(n+1)} = T^{(n)}$ since no more than n distinct rows can be fed into the process. Then $T^{(n)}$ represents all vertices that can be connected to v_i by paths of any length. If $T_i^{(n)} = 0$, v_i cannot be reached along a path from v_i; if $T_i^{(n)} = 1$, it can be reached. Hence

v_i lies on a cycle if and only if $T_i^{(n)} = 1$. The algorithm constructs the $T^{(j)}$ in a somewhat different fashion and it allows the process to terminate at any time that a 1 appears in the ith place of the vector T.

The algorithm itself allows one to feed in a given row no more than once. In a sense, it represents a cycle search with a memory of which vertices have already been reached.

In the application of this algorithm, a significant reduction can be made. The matrix α will represent the potential blocking graph after the granting of a request for a datum. As will become evident, we can assume that α is almost a strict upper triangular matrix; that is, $\alpha_{jk} = 0$ if $i \neq j \geq k$:

$\alpha =$ i



With this restriction on α, we can let T be a 1 x i row vector and use the $\hat{\alpha}_j = (\alpha_{j1}, \ldots, \alpha_{ji})$ for the α_j in the algorithm. This reduction is possible because for $k > j$, $v_k \not\rightarrow v_j$; if a cycle

containing v_i exists, that path does not contain any v_k where $k > i$ because each vertex following v_k in the circuit would be of the form v_{k+t}, $t > 0$.

SECTION III

UPPER TRIANGULAR MATRICES AND SAFE PERMUTATIONS

INTRODUCTION

This section will give theoretical justification for a matrix method of recording potential blocking in an implementation of the Scheduler. It is shown that if $\underline{P}$ is loop-free at time γ, then there is a permutation of the processes $\underline{P}$ such that the associated adjacency matrix is strictly upper triangular. This result is obtained using permutation matrices and elementary matrix operations. As mentioned at the end of Section II, recording the potential blocking graph as a strictly upper triangular matrix allows a definite reduction in algorithm β.

NOTATION AND BASIC FACTS

Let $\underline{P}_m$ denote the set of permutations on the set $\{1, \ldots, m\}$. If $\delta \in \underline{P}_m$ and $1 \leq j \leq m$, we will denote the image of j under δ by "$j\delta$". Let $\underline{M}_m$ denote the set of m x m matrices with m components equal to one (others 0), such that each row and each column has exactly one "1"-entry: I (the identity m x m matrix) is in $\underline{M}_m$. For $\delta \in \underline{P}_m$, let $M(\delta)$ be the unique matrix $(\alpha_{ij})_{1 \leq i, j \leq m}$ with

$$\alpha_{ij} = \begin{cases} 1, \text{ if } j = 1\delta \\ 0, \text{ otherwise .} \end{cases}$$

Clearly $M(\delta) \in \underline{M}_m$. Also, if δ_1 and δ_2 are distinct elements of $\underline{P}_m$, then $M(\delta_1)$ and $M(\delta_2)$ are distinct elements of $\underline{M}_m$. In addition, for $\alpha \in \underline{M}_m$, there is a (unique) $\delta \in \underline{P}_m$ such that $\alpha = M(\delta)$.

Consider $\delta_1, \delta_2 \in \underline{P}_m$. Write $M(\delta_1) = (\alpha_{ij})$ and $M(\delta_2) = (\beta_{ij})$. Let $(\gamma_{ij}) = (\alpha_{ij})(\beta_{ij})$. Clearly,

$$\gamma_{ij} = \begin{cases} 1, \text{ if } j = (i\delta_1)\delta_2 \\ 0, \text{ otherwise .} \end{cases}$$

Hence $(\gamma_{ij}) = M(\delta_1\delta_2)$ so that $M(\delta_1)M(\delta_2) = M(\delta_1\delta_2)$. We call $\underline{M}_m$ the set of m x m permutation matrices and note that $\underline{M}_m$ is in 1 - 1 correspondence to $\underline{P}_m$ by the correspondence $\delta \leftrightarrow M(\delta)$. In fact, one can show that $(X_1, \ldots, X_m)M(\delta) = (X_{1\delta}, \ldots, X_{m\delta})$.

MAIN DEVELOPMENT

Let the processes of a sequential machine be $P = \{P_1, \ldots, P_n\}$. Any $\delta \in \underline{P}_n$ generates a permutation $\{P_{1\delta}, \ldots, P_{n\delta}\}$ of the processes. For a state s of the machine and any $\delta \in \underline{P}_n$, there is an associated blocking graph $G(s, \delta)$. If $\delta' \in \underline{P}_n$, then $G(s, \delta)$ and $G(s, \delta')$ are isomorphic. Let the adjacency matrix of $G(s, \delta)$ be denoted $m(s, \delta)$. In general, $m(s, \delta)$ and $m(s, \delta')$ will not appear to be related. In fact, however, $m(s, \delta)$ and $m(s, \delta')$ are conjugates of a very special type.

Theorem 3.1: If $\delta, \delta' \in \underline{P}_n$, then $m(s, \delta) = M(\rho)^{-1} m(s, \delta')M(\rho)$ for some $\rho \in \underline{P}_n$.

Proof: Write $\delta = \delta'\rho$ and write $\rho = \prod_{k=1}^{t} \tau_k = \tau_1 \cdots \tau_t$, where each τ_k is a transposition. We induct on t:

t = 1: Let $t = (u, v)$. For any k and j between 1 and n,

$P_k \to P_j$ in state $s \Leftrightarrow P_{k\delta} \to P_{j\delta}$ in state s

$\Leftrightarrow P_{k\delta'} \to P_{j\delta'}$ in state s.

Thus, $m(s, \delta')_{k\delta', j\delta'} = m(s, \delta)_{k\delta, j\delta}$, where for any matrix a, a_{ij} denotes the (i, j)-component. To simplify notation, we write $k' = k\delta'$ and $j' = j\delta'$. Thus, $m(s, \delta')_{k', j'} = m(s, \delta)_{k'\rho, j'\rho}$.

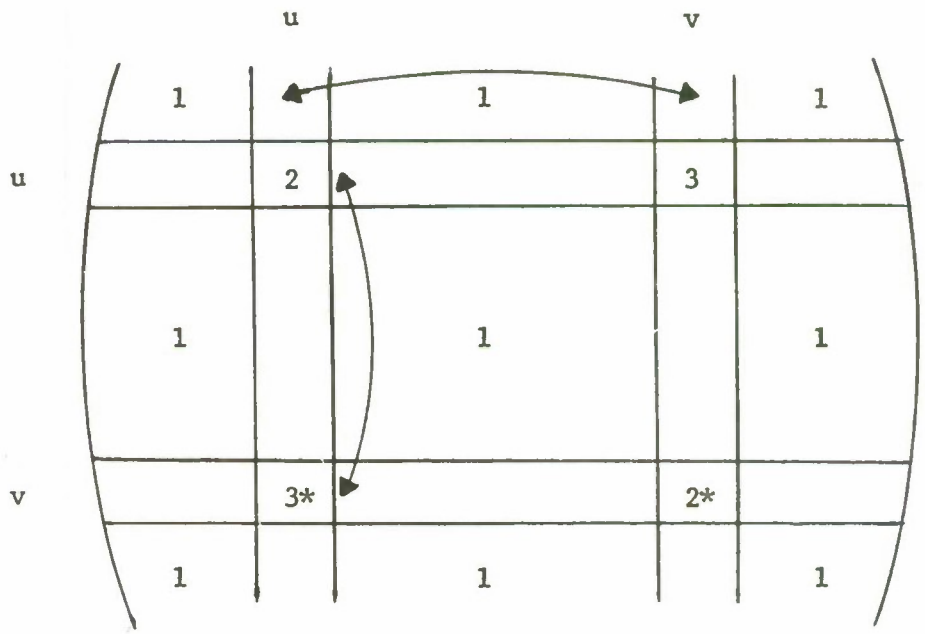
Case I: If $k', j' \notin \{u, v\}$, $k'\rho = k'$ and $j'\rho = j'$, so that $m(s, \delta')_{k', j'} = m(s, \delta)_{k', j'}$.

Case II: If $k' \notin \{u, v\}$ and $\{j', t\} = \{u, v\}$, then $k'\rho = k'$ and $j'\rho = t$. Thus, $m(s, \delta')_{k'u} = m(s, \delta')_{k'v}$ and $m(s, \delta')_{k,v} = m(s, \delta)_{k,u}$.

Case III: If $j' \notin \{u, v\}$ and $k' \in \{u, v\}$, then as in Case II, $m(s, \delta')_{uj'} = m(s, \delta)_{vj'}$ and $m(s, \delta')_{vj'} = m(s, \delta)_{uj'}$.

Case IV: If $\{k', r\} = \{u, v\} = \{j', t\}$, then $k'\rho = r$ and $j'\rho = t$. Thus, we have

$$m(s, \delta')_{uv} = m(s, \delta)_{vu};$$
$$m(s, \delta')_{uu} = m(s, \delta)_{vv};$$
$$m(s, \delta')_{vu} = m(s, \delta)_{uv}; \text{ and}$$
$$m(s, \delta')_{vv} = m(s, \delta)_{uu}.$$



Hence, m(s, δ') and m(s, δ) agree in areas marked 1; the values in rows u and v are interchanged (except the corners marked 2, 2*, 3, and 3*); the values in columns u and v are interchanged (except the corners marked 2, 2*, 3, and 3*); and the values (2 and 2*) and (3 and 3*) are interchanged.

Right multiplication of m(s, δ') by M(ρ) interchanges the u and v columns of m(s, δ); left multiplication of m(s, δ')M(ρ) by $M(\rho)^{-1} = M(\rho^{-1}) = M(\rho)$ interchanges the u and v rows. Hence, the description above of m(s, δ) in terms of m(s, δ') applies equally well to $M(\rho)^{-1}$m(s, δ)M(ρ). Hence, the equality holds.

Suppose the result holds when ρ can be expressed as a product of t - 1 transpositions, t > 1. Write $\delta = \delta'\rho'\tau$, where τ is a

transposition and ρ' can be expressed as a product of t - 1 transpositions. By assumption, $m(s, \delta'\rho') = M(\rho')^{-1} m(s, \delta')M(\rho')$; by the case $t = 1$, $m(s, \delta) = M(\tau)^{-1} m(s, \delta'\rho')M(\tau)$.

$$\begin{aligned} \text{Hence, } m(s, \delta) &= M(\tau)^{-1} M(\rho')^{-1} m(s, \delta')M(\rho')M(\tau) \\ &= \left[M(\rho')M(\tau)\right]^{-1} m(s, \delta')\, M(\rho')M(\tau) \\ &= M(\rho'\tau)^{-1} m(s, \delta')M(\rho'\tau) \ . \end{aligned}$$

Note: The permutation ρ is unique: $\rho = \delta(\delta')^{-1}$.

For a state s, let $m(s) = \{m(s, \delta): \ \delta \in \underline{P}_n\}$.

Theorem 3.2: If, for a state s, $\underline{P} = \{P_1, \ldots, P_n\}$ is loop-free, then

(*) m(s) contains a strictly upper triangular matrix (that is, a matrix which has zero entries on and below the main diagonal).

Proof: Let $\{P_1', \ldots, P_n'\}$ be the safe permutation generated by Theorem 5 in HC1. Then by Theorem 5 for each $k \in \{1, \ldots, n\}$

(2) $P_j' \not\rightarrow P_k'$ for each $j \in \{k+1, \ldots, n\}$ is satisfied. We consider $\delta \in \underline{P}_n$ such that $P_k' = P_{k\delta}$ for $k \in \{1, \ldots, n\}$. If $1 \le i < j \le n$, condition (2) says that $m(s, \delta)_{ji} = 0$. Since $P_k' \neq P_k'$ always, $m(s, \delta)_{kk} = 0$ for $k \in \{1, \ldots, n\}$. Hence, m(s δ) is a strictly upper triangular matrix.

Corollary 3.3: For a state s, $\underline{P}$ is loop-free if and only if condition (*) holds.

Proof: If $m(s, \delta)$ is strictly upper triangular, $\underline{P}$ must be loop-free: if $P_{i_1}, P_{i_2}, \ldots, P_{i_\rho}$ is a loop in $\underline{P}$, $i_1 < i_2 \ldots < i_\rho < i_1$, a contradiction.

Remark: Corollary 3.3 can be found in a context-free form in [1, p. 269].

SECTION IV

ALGORITHM γ

INTRODUCTION

Algorithm γ provides a systematic method of generating a safe permutation of the processes from an arbitrary permutation π when the set $\underline{P}$ of processes is loop-free. It also supplies a way to alter the potential blocking matrix so that it will correspond with the current safe permutation and thus preserve the advantages of recording the potential blocking matrix in strict upper triangular form.

Algorithm γ uses elementary matrix operations which can be performed routinely without actual matrix multiplication. The correctness and effectiveness of algorithm γ is proved using matrix arguments while its implementation uses the more prosaic matrix manipulation techniques described in this section.

DEVELOPMENT

The effectiveness of the algorithm will be established by repeated use of the following lemma:

Lemma 4.1: Let G be a digraph with vertex set $V = \{v_1, \ldots, v_n\}$. Let the $n \times n$ 0 - 1 matrix A_1 be the adjacency matrix of G. Suppose there is a sequence $\{a_1, \ldots, a_n\}$ of m distinct positive integers $\leq n$ such that $(A_k)_{a_1, a_{k+1}} = 1$ for

$1 \le k < n$ and $(A_n)_{a_1,a_2} = 1$ where the matrices A_k are defined recursively by $A_{k+1} = M((a_1, a_{k+1}))A_k M((a_1, a_{k+1}))$. Then the arcs (v_{a_1}, v_{a_2}), (v_{a_2}, v_{a_3}), . . . , $(v_{a_{n-1}}, v_{a_n})$, (v_{a_n}, v_{a_1}) form a closed path in G.

Proof: By the definition of A_{k+1}, the components of A_{k+1} and A_k have the following relations:

$$(a) \quad (A_{k+1})_{ij} = \begin{cases} (A_k)_{ij} \text{ if } i,j \notin \{a_1, a_{k+1}\} \\ (A_k)_{sj} \text{ if } j \notin \{a_1, a_{k+1}\} = \{i, s\} \\ (A_k)_{it} \text{ if } i \notin \{a_1, a_{k+1}\} = \{j, t\} \\ (A_k)_{st} \text{ if } \{a_1, a_{k+1}\} = \{i, s\} = \{j, t\} . \end{cases}$$

The following statement will be established by induction:

For $1 \le k \le n$, conditions (b), (c) and (d) below hold for A_k and the vertices V ordered relative to the permutation $(a_1, \ldots, a_k)$:

(b) if $i \notin \{a_s: \ 1 \le s \le k\}$, the ith row (column) represents the arcs of G beginning (ending) at V_i;

(c) if $1 \le i \le k$, the a_{i+1}-st row (column) represents the arcs of G beginning (ending) at V_{a_i}; and

(d) the a_1-st row (column) represents the arcs of G beginning (ending) at V_{a_k}.

The case $k = 1$ is trivially true by the definition of A_1.

Suppose the statement is true for u where $1 \le u \le n$.

Let $i \notin \{a_1, \ldots, a_{u+1}\}$. By conditions (a), (b) and (c) for u, the $\underline{i}$th row (column) of A_{u+1} satisfies condition (b) for u + 1, except perhaps at positions a_1 and a_{u+1}. Now $(A_{u+1})_{i,a_1} = (A_u)_{i,a_{u+1}}$ and $(A_{u+1})_{i,a_{u+1}} = (A_u)_{i,a_1}$. Since $(a_1, \ldots, a_{u+1})$ $= (a_1, \ldots, a_u)\ (a_1, a_{u+1})$, it follows that the equations in the preceding sentence are justified by the fact that the first equality means $(A_{u+1})_{i,a_1} = 1 \Leftrightarrow (v_i, v_{a_{u+1}})$ is an arc in $G \Leftrightarrow (A_u)_{i,a_{u+1}} = 1$ and the second equality means $(A_{u+1})_{i,a_{u+1}} = 1$. Hence, condition (b) is satisfied for u+1, and condition (c) is satisfied for u+1 and $1 \leq i < u$.

Now the a_1-st row (column) of A_{u+1} represents what the a_{u+1}-st row (column) of A_u represents, namely the arcs of G beginning (ending) at $v_{a_{u+1}}$. Similarly, the a_{u+1}-st row (column) of A_{u+1} represents what the a_1-st row of A_u represents, namely the arcs of G beginning (ending) at v_{a_u}. Hence, condition (c) is satisfied for u+1 and i=u and condition (d) is satisfied for u+1.

The statement is now proved.

The hypothesis $(A_k)_{a_1,a_{k+1}} = 1$ for $1 \leq k < n$ implies that there is an arc from v_{a_k} to $v_{a_{k+1}}$ in G. The hypothesis $(A_n)_{a_1,a_2} = 1$ implies that there is an arc from v_{a_u} to v_{a_1}. These arcs are the closed path of the lemma.

ALGORITHM γ

Let the state of the machine be s_ν. For any permutation π of the first n integers, when $\underline{P} = \{P_1, \ldots, P_n\}$, let $m(s_\nu, \pi)$ be the potential blocking matrix relative to the permutation π: the (i, j)-component of $m(s_\nu, \pi)$ is 1 if $P_{i\pi} \rightarrow P_{j\pi}$ in state s_ν and is 0 otherwise.

Let π be any permutation on $\{1, 2, \ldots, n\}$ and suppose $\underline{P}$ is loop-free in state s_ν. Then the potential blocking graph of $\underline{P}$ at state s_ν has no closed paths. Set $\alpha = m(s_\nu, \pi)$.

Step 1: Find the greatest s such that $\alpha_{su} = 1$ for some $u < s$.

Step 2: Find the least $t < s$ such that $a_{st} = 1$.

Step 3: Replace α by $M(s, t)\alpha M(s, t)$ and π by $\pi(s, t)$.

After Step 3, repeat Step 2, if possible. If Step 2 is not possible, repeat Step 1, if possible. (If it is possible to repeat Step 1, the new s' will be smaller than s, as will be explained.) If Step 1 is not possible at any time, α is a strictly upper triangular matrix, or equivalently, π is a safe permutation.

ANALYSIS OF THE ALGORITHM

After Step 1, we know that $\alpha_{ij} = 0$ for $i > s$ and after Step 2 that $\alpha_{sj} = 0$ for $j < t$. Since Step 3 interchanges the s and t rows and columns of α and the s and t columns are 0 below the s row, the new α also has 0 below the s row. Suppose after m repetitions of Step 2 are done for a given s of Step 1 and a given t of Step 2,

there is a 1 in the (s, t) position of α. The algorithm's operation makes lemma 4.1 applicable, so that there would be a closed path including $v_{s\pi}$ in the potential blocking graph in state s_ν. That is, $\underline{F}$ would have a loop, contrary to assumption. Hence, the algorithm can clear α of 1's in the s-row without introducing 1's below the s-row and under the main diagonal. Clearly, at most s - t repetitions of Step 2 with this s will clear row s of 1's and at most n-1 repetitions of Step 1 would clear all rows of 1's below the main diagonal.

SECTION V

A SCOREKEEPER FOR HARMONIOUS COOPERATION

INTRODUCTION

The operation of the Scheduler in HC1 depends on an algorithm which processes a request for a datum and responds in the following way:

1) it determines whether granting the request would cause conflict, and, if so, the process is added to the Q queue[1];
2) if granting the request would not cause conflict, it also generates the potential-blocking vector of the requesting process P_i.

In addition, an algorithm to process releases of data and the beginnings and ends of process runs is necessary. Both algorithms are encompassed in the Scorekeeper algorithm described below.

DESCRIPTION OF THE SCORECARD

The Scorekeeper can record both the current allocation of data and the future claims on the data base in an n x m x 4 matrix and an n x 2 matrix, both with entries of 0 and 1. The n x m x 4 matrix will be called the UVQJ matrix, the n x 2 matrix will be called the K-matrix (after the notation in HC1), and the aggregate will be called the Scorecard.

---

[1] As described in HC1.

A. The UVQJ Matrix

The UVQJ matrix will be considered to be 4 n x m matrices, referred to as the Use layer, the V layer, the Q layer, and the J layer, the letters corresponding to the notation in HC1 for the prescribed limits on each process's data needs. The (i, j)-component of the V (Q, J) layer is 1 if process P_i has included datum S_j in $V_i(Q_i, J_i)$ and is 0 otherwise. Since V, Q, and J are pairwise disjoint, at most one of those layers will have a 1-entry in any given (i, j)-component. If P_i is using datum S_j at state s_ν, the Use layer will have a 1 in the (i, j)-component; otherwise, the (i, j)-component is 0. The Use layer will be relatively active as the machine runs, while the remaining layers will remain relatively static. In fact, the V, Q, and J layers will change only when a process begins or ends its activity or when a process relinquishes its claim on a given datum. Figure 1 gives a visualization of the UVQJ matrix.

B. K-matrix

The K-matrix records the elements of elements of S in use during state s_ν: if P_i is using $\alpha_{ju} \in S_j$ during s_ν, row i will have the duple (j, u); if P_i is not using anything in inquiry-use mode, row i will have the duple (0, 0).

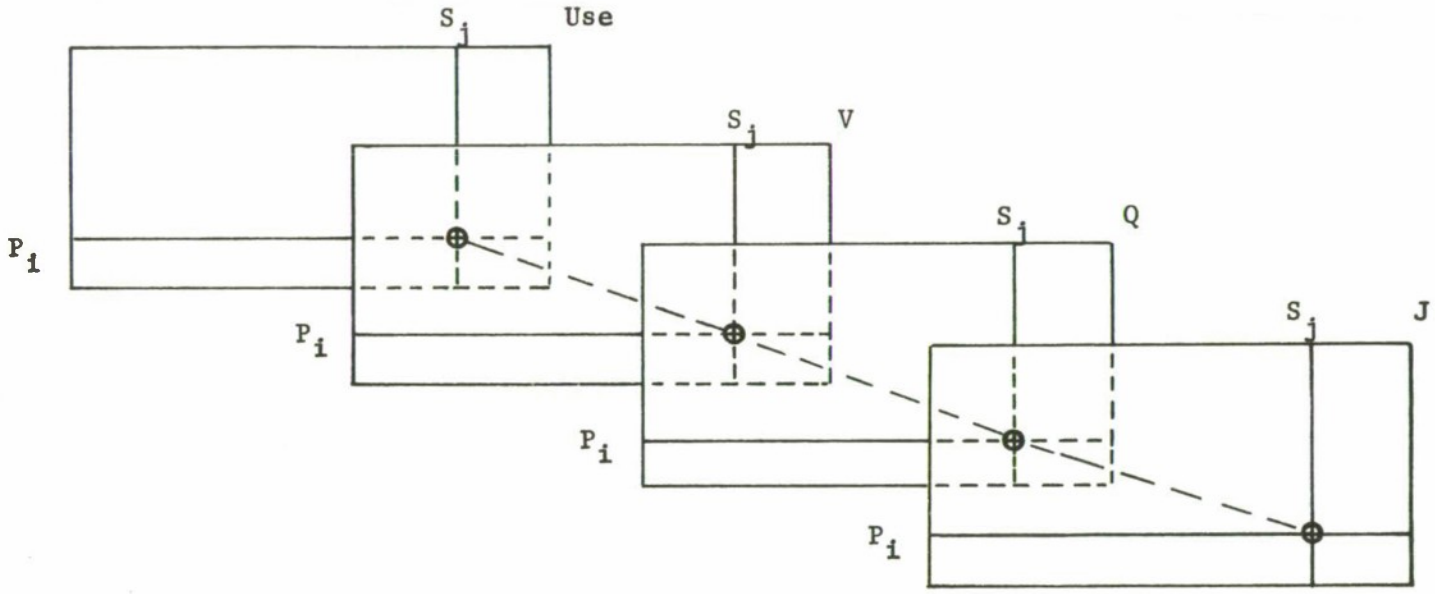


Figure 1. The UVQJ Matrix

USE OF THE SCOREKEEPER

The use of the Scorekeeper will be divided into nine parts--beginning a process's run, requests for an element of S in the write, read, or inquiry-use mode, the request for a subelement (that is, an element of an element of S), the release of an element of S, the release of a subelement, a reduction of future claims, and the end of a process's run.

In the course of this section, two operations (which are as yet undefined) will appear a number of times. The first operation determines whether the granting of a request would result in a safe permutation. The input of this operation is a 1 x n vector C whose entries are 0 or 1 and the output could be described as the set {safe, not-safe}. We shall denote this operation by "SAFETY(C)", and we shall describe it in more detail in Section VI. The second operation generates a new safe permutation and alters the potential blocking matrix α to match the new permutation. We shall denote this operation "UPDATE", and we shall present a full description in Section VI.

Another operation which occurs regularly in this section involves a determination of those indices k such that $k = j$ or that $S_k \rightarrow S_j$. This determination is necessary to check for conflict and to generate the potential-blocking vector which would result from granting a

specific request. We will let INDEX(j) be defined as the set $\{k: k = j \text{ or } S_k \rightarrow S_j\}$.[2]

The remainder of this section is devoted to a description of the Scorekeeper algorithm in the nine basic situations. The description of the action taken in each situation will begin with a précis of the action required by the third model of HCl. This is followed by a description of the action taken by the Scorekeeper algorithm juxtaposed with a diagram of that action.

1. P_i begins its run and declares its claim lists V_i, Q_i, J_i:

This action will alter the V, Q, and J layers and it will cause an alteration of the potential blocking matrix α. In particular, the lists Q_i, V_i, and J_i are entered in the Q, V, and J layers. Then for each process P_k it is determined whether P_k potentially blocks P_i: if P_k potentially blocks P_i, α_{A_k,A_i} is set equal to 1. The result of the preceding operation is (possibly) to add 1's to the A_i-column of α. Thus, the last operation that needs to be performed is UPDATE to restore α to strict upper triangularity. Figure 2 describes the Scorekeeper's action in this situation.

After the claim lists are entered, the Use row for each process P_k is checked. If $Use_{kj} = 1$, P_k is using S_j in some mode. A check of the (k, j)-components of the V, Q, and J layers will determine

[2]One should note that if the relation R, defined on elements of S, is equality, then INDEX(j) = {j} and that the Scorekeeper algorithm becomes greatly simplified.

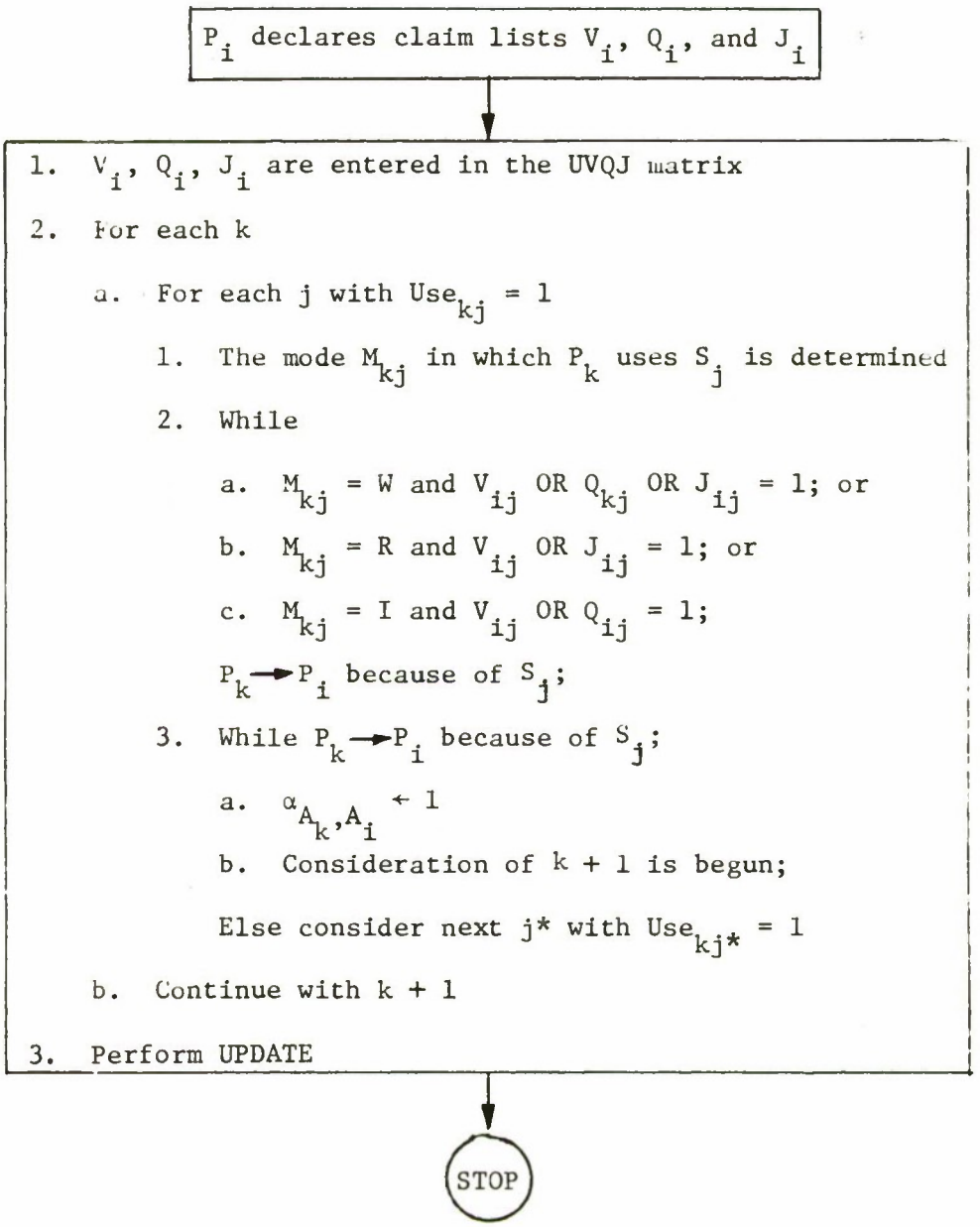


Figure 2. A Process Begins Its Run

the mode in which P_k is using S_j. If $V_{kj} = 1$, then P_k potentially blocks P_i if S_j is listed in any of P_i's claim lists (that is, if V_{ij} OR Q_{kj} OR $J_{ij} = 1$). If $V_{kj} = 0$ and $Q_{kj} = 1$, then P_k potentially blocks P_i if S_j is claimed in write or inquiry-use mode (that is, if V_{ij} OR $J_{ij} = 1$). If $V_{kj} = 0$ and $Q_{ij} = 0$, then (assuming as we are that P_k has access to S_j) $J_{kj} = 1$ and P_k potentially blocks P_i if S_j is claimed in write or read mode (that is, if V_{ij} OR $Q_{ij} = 1$). If it is determined that P_k does potentially block P_i, it is no longer necessary to check the rest of the data P_k is using. A 1 is entered in the (A_k, A_i)-component of α: process $P_k(P_i)$ is in position $A_k(A_i)$ in the current safe permutation A. The search then continues by considering the next process P_{k+1}. If P_k does not potentially block P_i, then consideration of P_{k+1} is begun. After all the processes have been considered, α may no longer be strictly upper triangular, and UPDATE is performed to rectify the situation.

2. P_i requests S_j in write mode:

Granting this request would result in conflict if any other process is using any S_k ($k \in$ INDEX(j)) in any mode. If granting the request would not result in conflict, P_i would potentially block any process that had S_k in its V, Q, or J list for any $k \in$ INDEX(j). Figure 3 describes the investigation of this request as carried out by the Scorekeeper.

If for any $k \in$ INDEX(j), the kth column of the Use layer has a 1 in it, the granting of P_i's request would cause conflict and

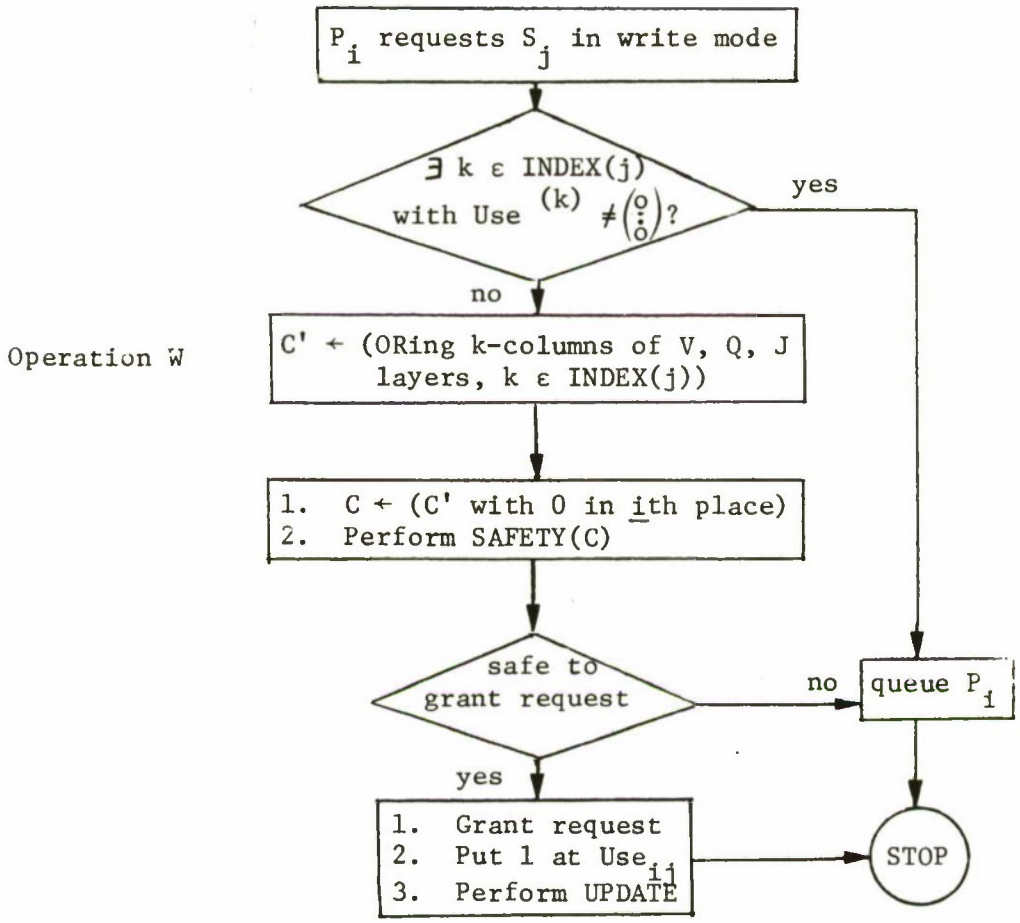


Figure 3. The Investigation of a Write Mode Request

P_i is queued. If no conflict would occur, the kth columns ($k \in$ INDEX(j)) of the V, Q, or J layers are ORed: this operation will be called Operation W. The result of Operation W is a vector C' whose 1's indicate which processes P_i would potentially block as a result of the granting of this particular request. Since $V_{ij} = 1$, the vector C' has a 1 in the ith place; this 1 is removed, creating a vector C and SAFETY(C) is performed. If SAFETY(C) determines that granting this request would yield a safe permutation, the request is granted, a 1 is put in the (i, j)-component of the Use layer to record P_i's current access to S_j, and UPDATE is performed.

3. P_i requests S_j in read mode:

Granting this request would result in conflict if any other process is using any S_k ($k \in$ INDEX(j)) in either write or inquiry-use mode. If granting the request would not result in conflict, P_i would potentially block any process that had any S_k ($k \in$ INDEX(j)) in its V or J list. Figure 4 describes the investigation of the request as carried out by the Scorekeeper.

If for any $k \in$ INDEX(j), the kth column of the Use layer has a 1-entry, then the determination of whether the granting of P_i's request would cause conflict can be made with one check. Suppose $Use_{uk} = 1$. If $Q_{uk} = 0$, P_u is using S_k in a mode other than the read mode and conflict would result if P_i's request were granted: P_i is queued. If $Q_{uk} = 1$, P_u is reading S_k. If $Use_{vk'} = 1$ where $k' \in$ INDEX(j), P_v must be reading $S_{k'}$, since otherwise there would

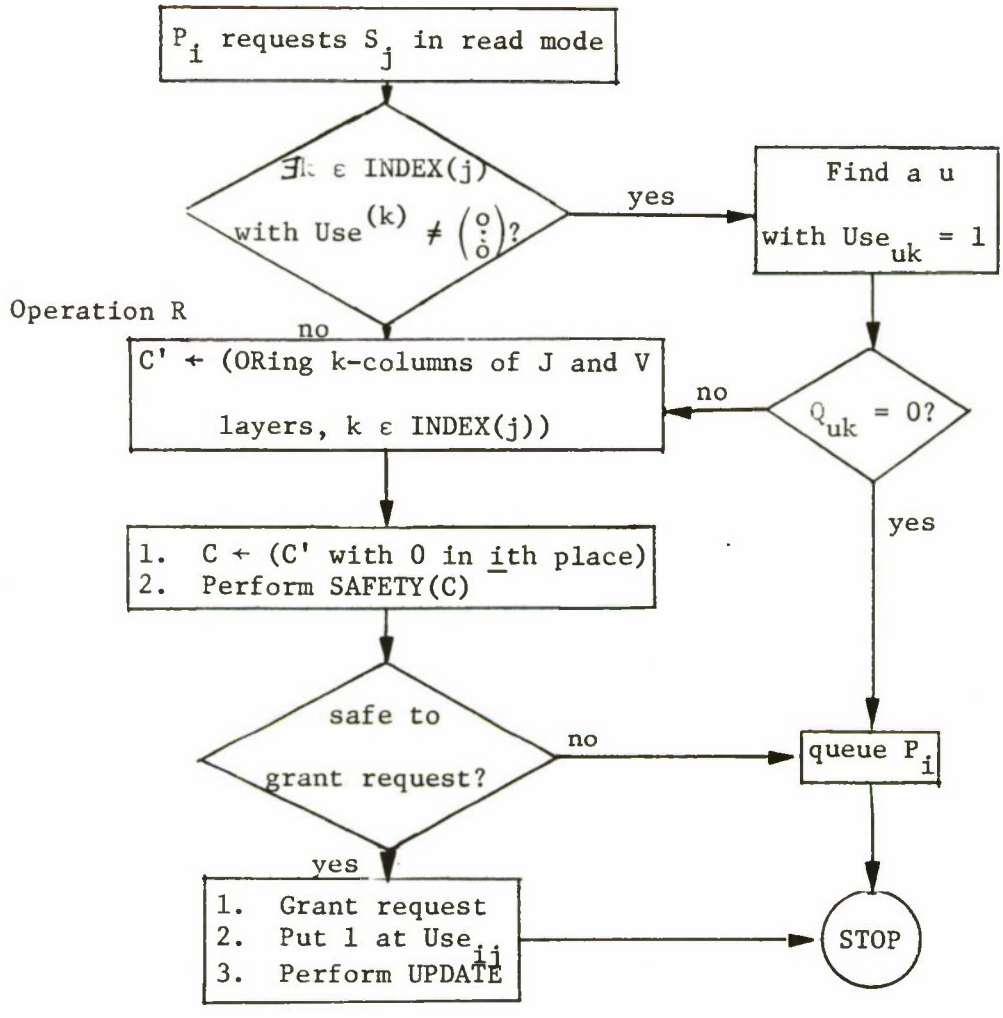


Figure 4. The Investigation of a Read Mode Request

have been conflict before P_i made his request and conflict is never allowed by the Scheduler. Hence, one check in the Q layer is sufficient to approve or disapprove P_i's request on the basis of conflict. If no conflict would ensue, the Operation R is performed: OR the k-columns (k ε INDEX(j)) of the J and V layers to produce a vector C'. As in 1 above, C' represents the potential blocking of P_i as a result of the granting of his request, except that a 1 appears in the ith position. The vector C is created from C' by putting a 0 in the ith place, and SAFETY(C) is performed. If a safe permutation would not result from granting this request, P_i is queued; if a safe permutation would result, P_i is granted his request, a 1 is put in the (i, j)-component of Use and UPDATE is performed.

4. P_i requests S_j in inquiry-use mode:

Granting this request would result in conflict if any other process is using any S_k (k ε INDEX(j)) in write or read modes. If granting the request does not result in conflict, P_i would potentially block any process that has S_k (k ε INDEX(j)) in its Q or V list. Figure 5 describes the investigation of this request carried out by the Scorekeeper.

If for any k ε INDEX(j) the kth column of the Use layer has a 1-entry, then a single look at the J layer will determine whether conflict will result. If $Use_{vk} = 1$, we look at J_{vk}. If $J_{vk} = 0$, P_v is using S_k in a mode other than inquiry-use and conflict would result from granting P_i's request: P_i is queued. If $J_{vk} = 1$, every

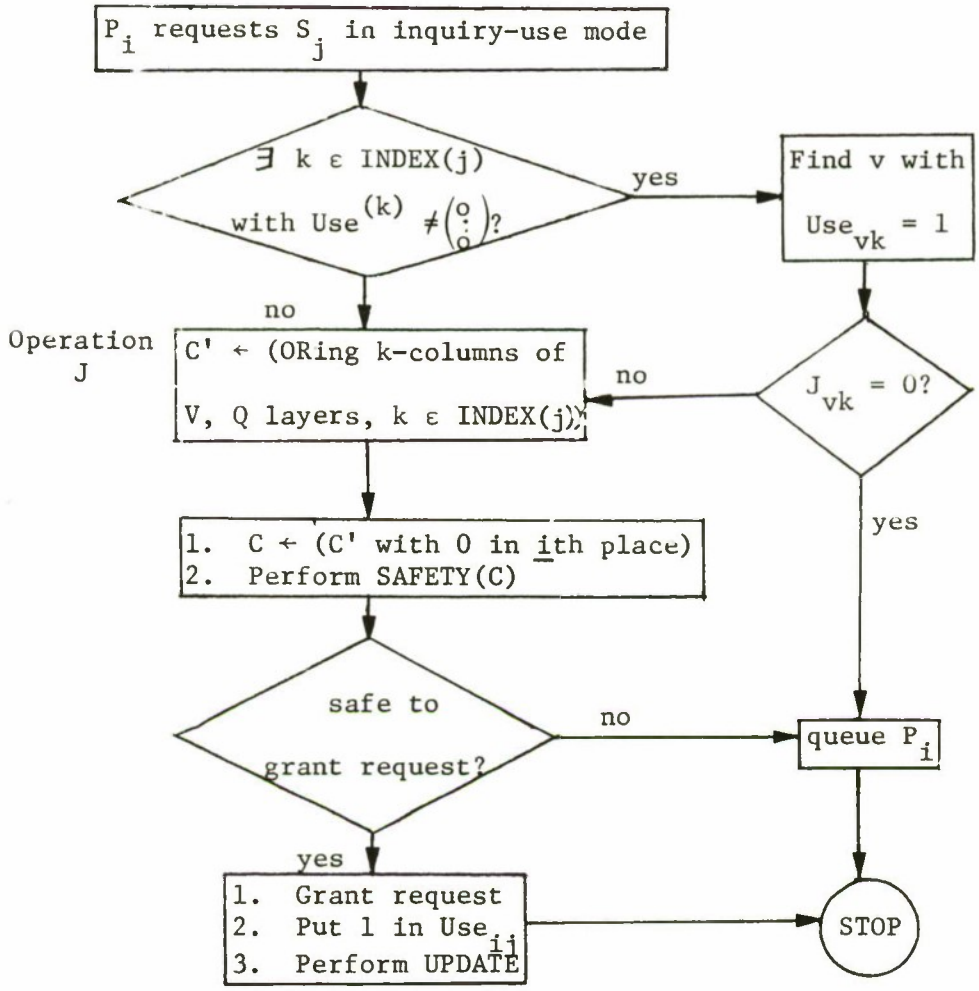


Figure 5. The Investigation of an Inquiry-Use Mode Request

process using any S_k, ($k \in$ INDEX(j)) is using S_k, in inquiry-use mode. Next, Operation J is performed: OR the k-columns ($k \in$ INDEX(j)) of the Q and V layers. As before, a potential blocking vector C' is created, C is made by putting a 0 in the ith position of C', and SAFETY(C) is performed. If the new permutation would not be safe, P_i is queued. If the permutation would be safe, P_i is granted his request, a 1 is put in Use_{ij}, and UPDATE is performed.

5. P_i requests $\alpha_{ju} \in S_j \in I_i$:

Granting this request would result in conflict if any other process is using a $\beta \in S_k$ ($k \in$ INDEX(j)) where $\beta \leftrightarrow \alpha_{ju}$. Since the definition of potential blocking does not include any consideration of the K_i, it is not necessary to generate a potential blocking vector C nor to perform SAFETY(C). Figure 6 describes the investigation of this request.

The set Γ of elements of elements of S related to α_{ju} by the relation $\leftrightarrow$ is determined. If (k, v) appears in the K-matrix where $\alpha_{kv} \in \Gamma$, conflict would result from granting this request so that P_i must be queued. If no such (k, v) appears in the K-matrix, the request is granted and the duple (j, u) is put into the ith row of the K-matrix.

6. P_i relinquishes access to S_j:

This action will leave the potential blocking matrix unchanged or will replace some 1's on row A_i with 0's. Figure 7 summarizes

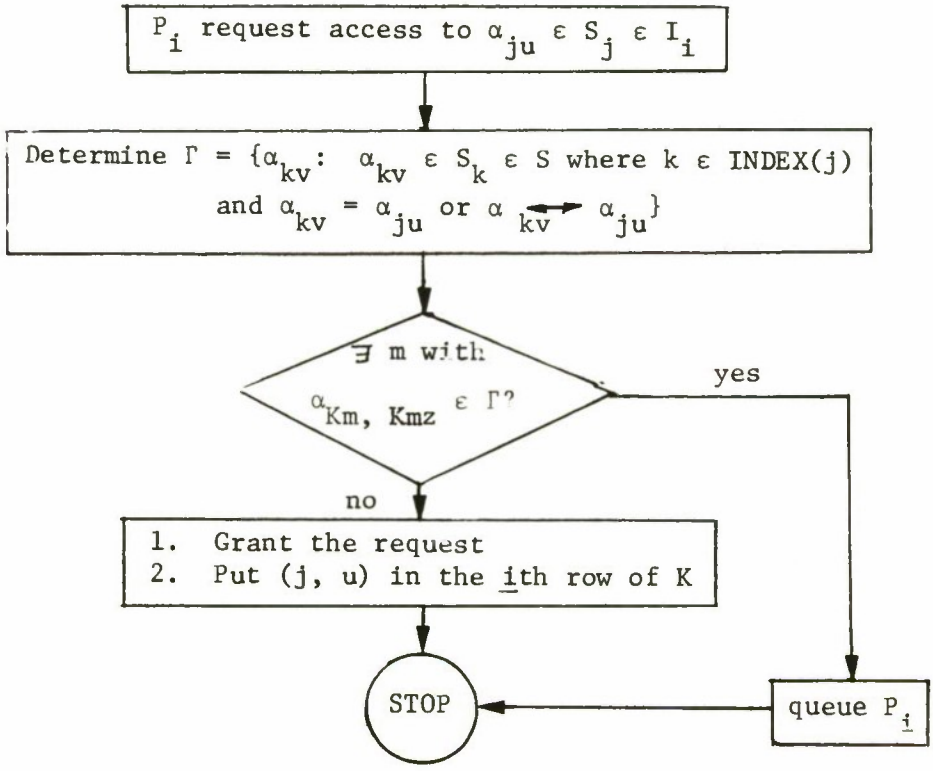


Figure 6. The Investigation of a Request for a Subelement

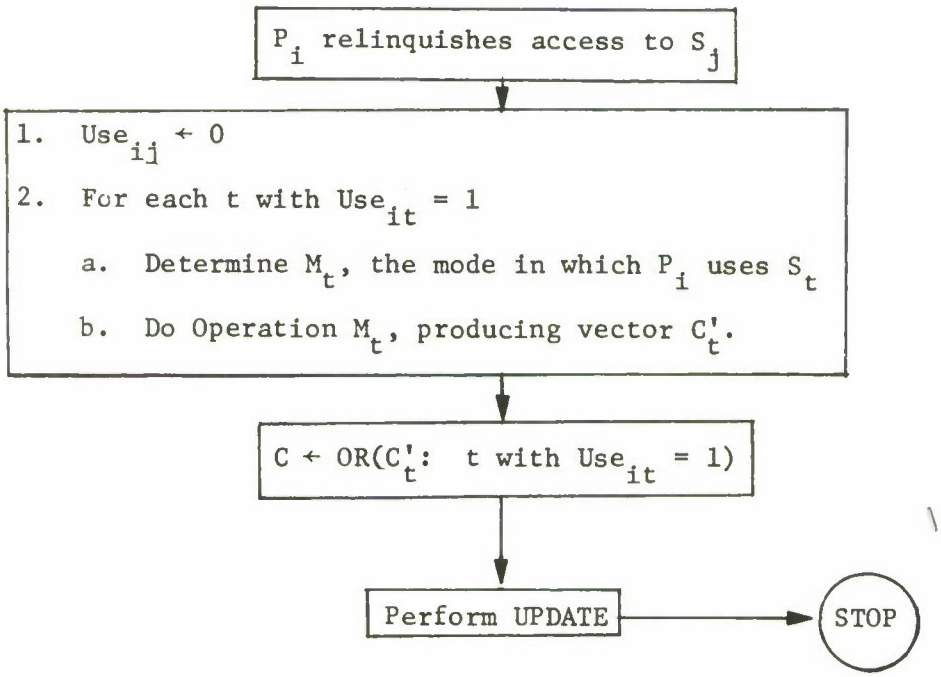


Figure 7. The Release of a Datum

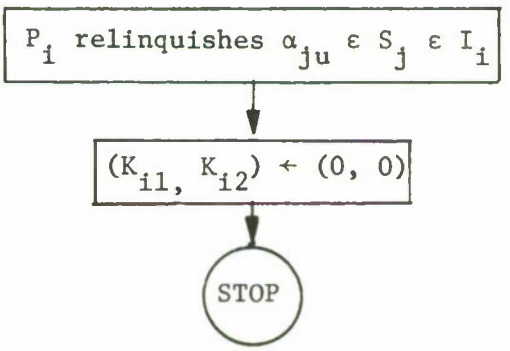


Figure 8. P_i Releases a Subelement

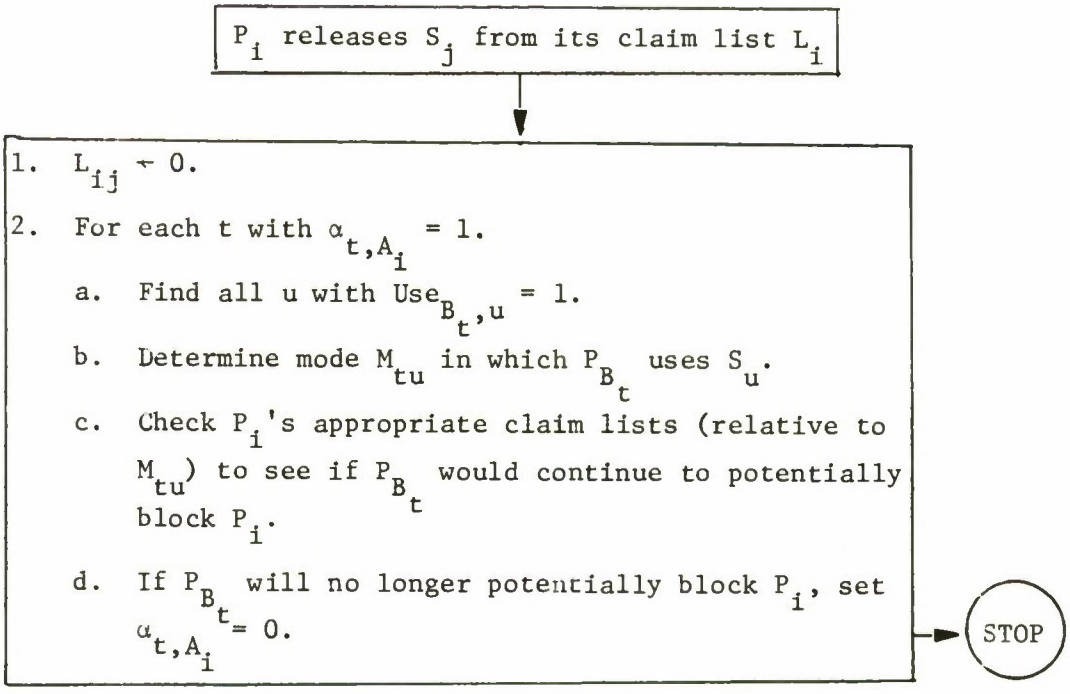


Figure 9. The Reduction of a Claim List

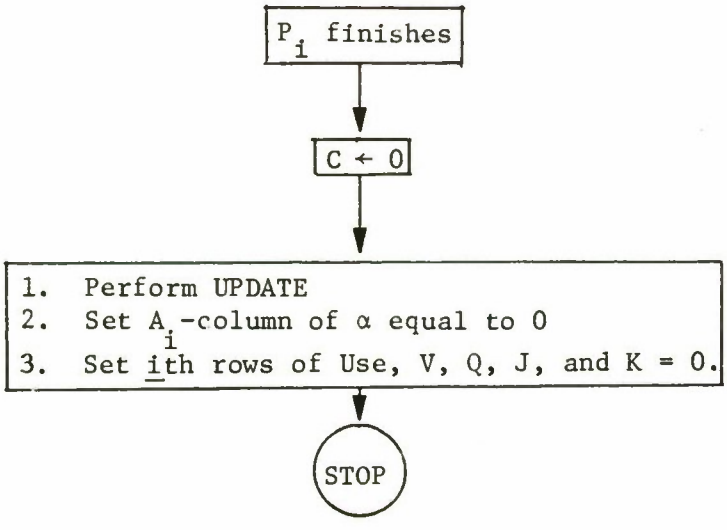


Figure 10. A Process Finishes

the action of the Scorekeeper.

The (i, j)-component of the Use layer will be set to 0 to indicate that P_i is not using S_j. The potential blocking vector of P_i now needs to be recalculated. This is done by applying the appropriate operation--W, R, or J--for each datum S_t that P_i is still using. The calculation of the blocking vector is completed and UPDATE is performed.

7. P_i relinquishes access to $\alpha_{ju} \in S_j \in I_i$:

Since the definition of potential blocking does not involve the contents of the K_i, P_i's relinquishing α_{ju} does not alter the potential blocking matrix. Hence, the only action necessary when P_i releases some subelement is to clear the ith row of the K-matrix. Figure 8 depicts this action.

8. P_i releases S_j from its claim list:

This action will leave the potential blocking matrix unchanged or will replace some 1's on column A_i with 0s. Figure 9 summarizes the action of the Scorekeeper in this situation.

If P_i wants to delete S_j from its limit list L_i (where $L \in \{V, Q, J\}$, the (i, j)-component of the L layer is set equal to 0. The column vector which indicates which processes potentially block P_i after this deletion is next calculated. If $A = (A_1, \ldots, A_n)$ is the current permutation such that $\alpha = m(s_\nu, A)$ and $B = A^{-1}$, one proceeds by finding processes P_{B_t}

who were potentially blocking P_i before the deletion and determining whether they potentially block P_i after the deletion. This is done by finding t with $\alpha_{t,A_i} = 1$ and then finding u with $Use_{B_t,u} = 1$ and then finding u with $Use_{B_t,u} = 1$. For every pair (t, u) of this type we set

$$\alpha_{t,A_i} = \begin{cases} OR(V_{ir},Q_{ir},J_{ir}: \ r \in INDEX(u), \ Use_{B_t,u} = 1) \text{ if } V_{B_t,u} = 1 \\ OR(V_{ir},J_{ir}: \ r \in INDEX(u), \ Use_{B_t,u} = 1) \text{ if } Q_{B_t,u} = 1 \\ OR(V_{ir},Q_{ir}: \ r \in INDEX(u), \ Use_{B_t,u} = 1) \text{ if } J_{B_t,u} = 1 \ . \end{cases}$$

That is, α_{t,A_i} is set equal to 1 if and only if P_i has placed a future claim on a datum $S_r \longleftrightarrow S_u$ and the claim is in a mode which could cause conflict with P_{B_t}.

9. P_i finishes:

This operation is the easiest to deal with. P_i is no longer potentially blocked, potentially blocks no one, and his Scorecard is 0. Figure 10 summarizes the Scorekeeper's action for this operation.

The A_i rows and columns of the potential blocking matrix α are filled with 0's, and the ith rows of Use, V, Q, J, and K are filled with 0's. The vector C is set equal to 0 because use is made of C in performing UPDATE.

SECTION VI

AN OVERVIEW OF THE OPERATION OF THE SCHEDULER

INTRODUCTION

In this section the use and interrelations of the algorithms in previous sections of this paper are described. In addition, the operations SAFETY(C) and UPDATE are specified. Finally, brief analysis of this implementation of HCl's model is given in terms of necessary storage space for the matrices and vectors of the algorithms.

DESCRIPTION OF THE OPERATION OF THE SCHEDULER

Assume that the Scheduler is to handle n processes concurrently and that the data base has m pieces of data. The plan of implementation presented in this paper requires the following apparatus:

1) an n x n potential blocking matrix α;
2) a 1 x n vector A for recording the current safe permutation and a 1 x n vector B to record the inverse of A;
3) two 1 x n vectors, T and E, for use in algorithm β for checking for a safe permutation;
4) the Scorecard, consisting of the n x m x 4 UVQJ matrix and the n x 2 K-array; and
5) a 1 x n vector C for use in calculating potential blocking vectors.

At time ν_0, A = B = (1, . . . , n) and every other vector or matrix is set equal to 0.

When a process P_i begins its run, it is put in queue T, the temporary-holding queue, if B, the special-attention queue, is non-empty and it is put into E, the set of running processes, if B is empty. In any case, the Scorekeeper performs the necessary bookkeeping to record P_i's claim lists and to correct the potential blocking matrix (as described in Section V).

When P_i requests a datum S_j, the request is processed as described in Section V. The first check is for conflict. If conflict would occur, the request is denied and P_i is queued on Q. If conflict would not occur, the vector C is generated (as described in Section V) which shows those processes P_i would potentially block because of his access to S_j if his request were granted. Then SAFETY(C) is performed, as described here. First, T is set equal to $OR(\alpha_{A_i}, (C_{A_1}, C_{A_2}, \ldots, C_{A_n}))$ where $C = (C_1, \ldots, C_n)$. This rearrangement is necessary because the Scorecard keeps records relative to the identity permutation I whereas the matrix α keeps records relative to the current safe permutation A. The A_i-th row of α is ORed with the rearranged vector C because C records only the potential blocking of P_i due to his possession of S_j. Algorithm β is run on T with the vector E keeping track of which rows have already been ORed into T (as described in Section II): when row k of α is ORed into T, a 1 is put in E_k. The determination in algorithm β whether there is a $j \notin E$ with $T_j \neq 0$ is made by sweeping T and at any 1 checking the same entry of E: if there is a 1, continue the sweep; if there is a 0, a 1 is added and the appropriate

row of α is ORed into T. If algorithm β determines that the resulting permutation would not be safe, SAFETY(C) = not-safe and P_i is queued. If SAFETY(C) = safe, P_i is granted his request for S_j, a 1 is put at Use_{ij} (as described in Section V), and UPDATE is performed. The vector C is ORed into the potential blocking matrix α, relative to the safe permutation A. That is, $\alpha_{A_i} = (\alpha_{A_i,1}, \ldots, \alpha_{A_i,n})$ and $(C_{A_1}, \ldots, C_{A_n})$ are ORed to create a new A_i-row for the matrix. The last action in UPDATE of the Scheduler is algorithm γ (described in Section IV). Matrix α may now no longer be a strictly upper triangular matrix. Each nonzero entry of α is removed by an interchange of a pair of rows and columns. If the ith and jth rows and columns are switched, then the values A_i and A_j are switched and the values B_{A_i} and B_{A_j} are switched. After this action A will still represent the "current" safe permutation so that $\alpha = m(s_{\nu+1}, A)$ and $B = A^{-1}$. It might be noted that the switch of B_{A_i} and B_{A_j} can be made either before or after the switch of A_i and A_j; the result is the same.

When a process releases a datum, the Scheduler alters the Scorecard and the matrix α as described in Section V. The algorithm in Section IV of HC1 concerning the release of a datum is performed next. In essence, the special priority queue B is checked and emptied if possible. If B is emptied, all the processes suspended in the temporary queue T are allowed to begin their run. If B cannot be emptied (or if it was empty to begin with), the queue Q

is swept in order. Each process in the queue has his last request reprocessed as outlined in the preceding paragraphs. If the element that had been requested is available (viz., if that element was just released) and the granting of the request is permissable, the request is granted and the process is removed from the queue Q. If the element is available but the granting of the request is not permissable, then the process is put in queue B, if it is empty, and all processes wanting to enter the system are queued in T. If there are several processes whose requested element is now available and whose request is still not permissable, then the first such process in Q will be put into queue B and the others will be left in queue Q.

The Scheduler's actions under a reduction of a claim list or an end of a process's run are treated in Section V. The Scorecard entries of the process are altered and the potential blocking matrix α is altered by recalculating the potential blocking (or potentially blocked) vector of that process and entering it in α and by performing UPDATE. In addition, after a process ends its run, a queue search, exactly as that described in the paragraph above, is initiated.

The preceding paragraphs provide a complete functional description of the operation of the Scheduler. Moreover, the entire operation of the Scheduler is encompassed in the algorithms of this paper as presented in this section.

ANALYSIS OF STORAGE REQUIREMENTS

The description of this section allows immediate calculation of the storage requirements for this realization of the model of HC1. It can be shown that the positive integer n can be expressed with N binary digits, where $N = [\log_2 n] + \delta$, $\delta = 0$ or 1. Similarly, set $M = [\log_2 m] + \gamma$, $\gamma = 0$ or 1. The various parts of the Scheduler algorithms then require the following amounts of storage:

the matrix α	n^2 bits;
the vectors A and B	nN bits each;
the vectors C, T, and E	n bits each
the UVQJ matrix	4nm bits; and
the K-array	n(M+k) bits,

where an identification for an element of an element of the data base requires no more than k bits. Setting $k = [\log_2 j] + \epsilon$, $\epsilon = 0$ or 1 where j is the maximum number of elements in an element of S will usually be sufficient. The total necessary storage is $n^2 + n(2N + 4m + M + k + 3)$; of this total, only $n^2 + n(2N + 3m + 3)$ bits are required if the inquiry-use mode is eliminated from the model, as in Section V of HC1.

As an example, suppose a 256K word machine with 16-bit words is to be used to provide data-sharing for 64 concurrent users of 1000 files, each with no more than 64,000 records. Since 64,000 can be expressed with 16 binary digits, we can set k = 16. With $n = 64 = 2^6$, N = 6. Further, m = 1000 so that the Scheduler would require

262,720 bits of storage or 16,420 words. On a 256K machine, this is an overhead of about 6.42% of the total storage. If the inquiry use mode were omitted, only 197,056 bits = 12,316 words are needed, a 4.72% overhead. If the number of files were only 500, the respective overhead figures would be 3.28% and 2.46%.

It should be kept in mind that the figures above do not take into account the storage space necessary for (i) the programs which actualize the algorithms and (ii) the space necessary for the indexing necessary for the matrices and vectors involved. The storage estimates include only the space necessary for the bulk information required by this method of realizing HCl's model.

CONCLUDING REMARKS

This paper has presented one particular realization of the Scheduler of HCl. A set of record-keeping devices and algorithms for their use have been suggested and a dynamic description of the functioning Scheduler has been provided. In addition, full theoretical justification for the algorithms has been presented, showing both that the algorithms are correct and that they are actually realizable (in the sense that every algorithm will indeed terminate).

It should be recognized that the important determination of the sets INDEX(j) has been everywhere ignored. This determination is directly dependent on the relations R and R', which relations are

very system-dependent. Since the nature of R and R' and thus the determination of the sets INDEX(j) will always be idiosyncratic to a particular situation, it was decided that a general treatment of that problem would not be presented in this paper.

The analysis of storage requirements in Section VI provides not only a measure of the space needed to implement the Scheduler in a given data-sharing situation, but also a rough numerical gauge for comparing the model of HCl as realized in this paper with other data-sharing models which may be under consideration. In a direct way, therefore, this paper is a first tangible (and necessary) step towards the tradeoff analyses of data-sharing techniques which are part of the specified End Products of Project 6710 (3.3.2).

BIBLIOGRAPHY

1. Harary, Frank, Norman, Robert Z., and Cartwright, Dorwin. Structured Models: An Introduction to the Theory of Directed Graphs. New York: John Wiley and Sons, Inc., 1965.

2. LaPadula, Leonard J. "Harmonious Cooperation of Processes Operating on a Common Set of Data, Part 1." ESD-TR-72-147, Vol. 1.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
The MITRE Corporation P. O. Box 208 Bedford, Massachusetts 01730	UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

HARMONIOUS COOPERATION OF PROCESSES OPERATING ON A COMMON SET OF DATA, PART 2

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (First name, middle initial, last name)

D. Elliott Bell

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
DECEMBER 1972	50	2

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
F19628-71-C-0002	
b. PROJECT NO. 671A	ESD-TR-72-147, Vol. 2
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	MTR-2254, Vol. II

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Deputy for Command & Management Systems Electronic Systems Division, AFSC L. G. Hanscom Field, Bedford, Mass. 017?

13. ABSTRACT

This report provides algorithms (and proofs of their correctness) which actualize the data-sharing model of "Harmonious Cooperation of Processes Operating on a Common Set of Data, Part 1" (MTR-2254). Also included are a sketch of the coordination of the algorithms in Part 2 in the operation of the Scheduler of Part 1 and a brief analysis of the storage required to implement this version of the Scheduler.

DD FORM 1 NOV 65 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
DATA SHARING						
DEADLOCK						
FINITE-STATE MACHINES						
PERMANENT BLOCKING						